Лабораторная работа № 7

Тема: Построение 3D – объектов с учетом освещения

Задание:

Создать приложение Windows для изображения усеченной пирамиды, которая освещается источником света.

Изменяемые параметры:

1. положение источника света в мировой сферической системе координат $(r = \infty, \varphi, \theta)$;
2. положение наблюдателя в мировой сферической системе координат (r, φ, θ);
3. цвет источника света;

Использовать аксонометрическую проекцию фигуры на картинную плоскость.

Использовать диффузионную модель отражения света от граней пирамиды.

Обеспечить изображение фигуры при перемещении ***источника света*** по углу φ (клавиши «→» и «←») и углу θ (клавиши «↑» и «↓») ***при фиксированном положении наблюдателя.***

Обеспечить масштабирование фигуры при изменении размеров окна.

Для решения задачи дополнить класс CPyramid функцией:

void CPyramid::ColorDraw(CDC& dc, CMatrix& PView, CMatrix& PLight
CRect& RW,COLORREF Color)

```
// Рисует пирамиду, освещенную источником заданного цвета
// Самостоятельный пересчет координат из мировых в оконные (MM_TEXT)
// dc - ссылка на класс CDC MFC
// PView - координаты точки наблюдения в мировой сферической системе
 //координат (r, fi(град.), q(град.))
// PLight - координаты источника света в мировой сферической системе
// координат (r, fi(град.), q(град.))
// RW - область в окне для отображения
// Color – цвет источника
```

Теоретические сведения:

Сферическая система координат.

Сферическая система координат – трехмерная система координат, в которой каждая точка пространства определяется тремя числами (r, θ, ϕ), где r — расстояние до начала координат (радиальное расстояние), а θ и ϕ — зенитный и азимутальный углы соответственно рис. 1.

Зенит — направление вертикального подъёма над произвольно выбранной точкой (точкой наблюдения), принадлежащей фундаментальной плоскости.

Азимут — угол между произвольно выбранным лучом фундаментальной плоскости с началом в точке наблюдения и другим лучом этой плоскости, имеющим общее начало с первым.

Положение точки P в сферической системе координат определяется тройкой , где:

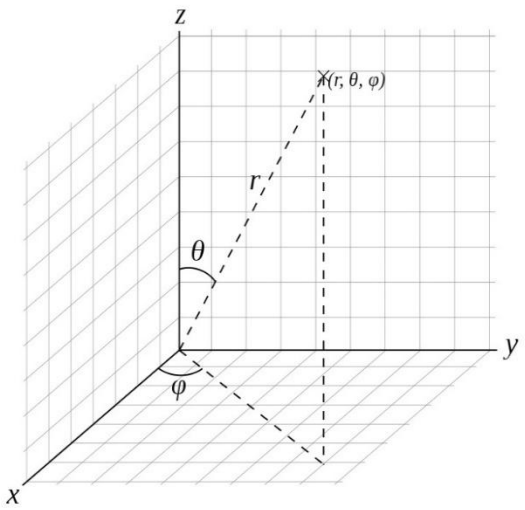


Рисунок 1- Сферическая система координат

$r \geq 0$ — расстояние от начала координат до заданной точки

$0° \leq \theta \leq 180°$ — угол между осью **z** и отрезком, соединяющим начало координат и точку P.

$0° \leq \varphi \leq 360°$

Однородные координаты

Однородные координаты – это математический механизм, связанный с определением положения точек в пространстве. Привычный аппарат декартовых координат, не подходит для решения некоторых важных задач в силу следующих соображений:

1. в декартовых невозможно описать бесконечно удаленную точку.
2. не позволяет произвести проверку различия между точкой и вектором.
3. невозможно использовать унифицированный механизм работы с матрицами для выражения преобразований точек.
4. декартовы координаты не позволяют использовать запись для создания перспективного преобразования точку.

Однородные координаты в двумерном изображении имеют вид (x, y, w), где w-масштабный множитель.

Двумерные декартовы координаты точки получаются из однородных делением на множитель w:

$$X=x/w,\ Y=y/w$$

Основные свойства однородных координат:

1. наборы чисел однородных координат могут соответствовать одной точке в Декартовых координатах, например точки: (6, 8, 4) и (3, 4, 2)
2. в силу произвольного значения **w** не существует единственного представления точки, заданной в декартовых координатах
3. как минимум одно число из трех, не должно равняться 0
4. деление на **w** всех координат даст Декартовы координаты (x/w, y/w,1)
5. при w=0, точка находится в бесконечности

Все матрицы преобразований имеют размер 3x3. Таким образом матрицы преобразования имеют один и тоже же вид.

Матрица переноса:

p’ = p T(Dx,Dy), где:

p’ – новые координаты

p – старые координаты

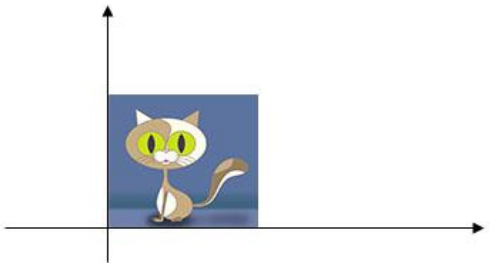


Рисунок 2- Изображение в начале координат

$$T(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

Рисунок 3- Результат переноса

$$[x',y',1] = [x, y, 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

Матрица растяжение (сжатия):

x’ = αx,α>0,

y’ = βy,β>0.

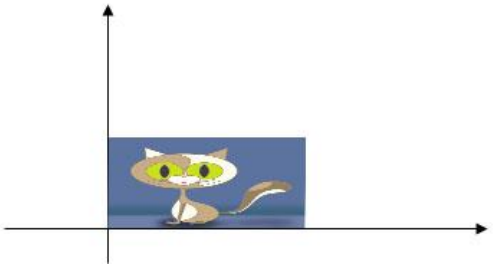


Рисунок 4- результат работы растяжения/сжатия

$$Taf = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матрица вращения относительно центра:

x’ = xcosϕ - ysinϕ,

y’ = xsinϕ – ycosϕ,

$$Taf = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

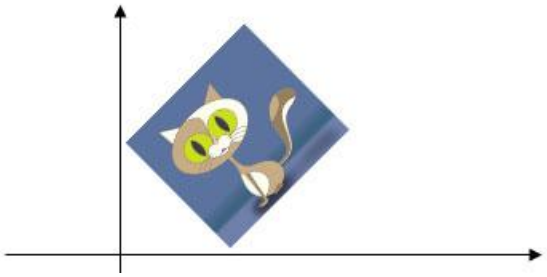


Рисунок 5- результат работы вращения

Матрица отражения:

x’ = x (x’ = -x),

y’ = -y(y’ = y),

$$Taf = \begin{bmatrix} 1 & 0 & 0 \\ \sin\varphi & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ - ось OY,

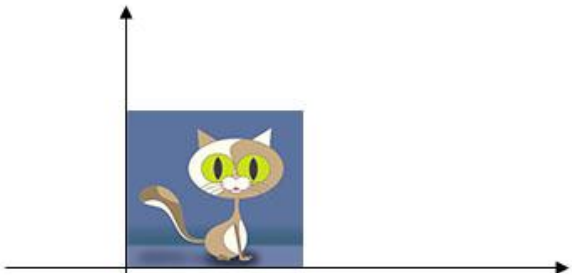


Рисунок 6- отражение по оси OY

$$Taf = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ – по оси OX

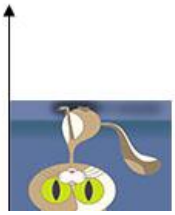


Рисунок 7-отражение по оси OX

Видовая система координат

Для изображения объекта на экране нужно пересчитать его мировые координаты в другую системы координат, которая связана с точкой наблюдения. Эта система координат называется видовой системой координат и является левосторонней.

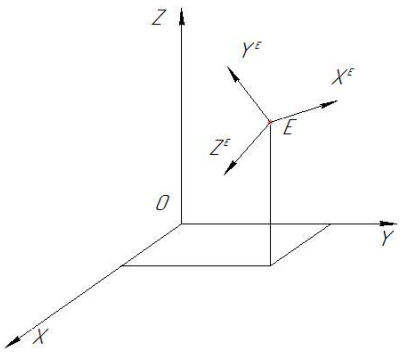


Рисунок 8- Видовая система координат

Диффузионную модель отражения света

Диффузное отражение – это вид отражения присущ матовым поверхностям . Матовой можно считать такую поверхность, размер шероховатостей которой уже настолько велик, что падающий луч рассеивается равномерно во все стороны.

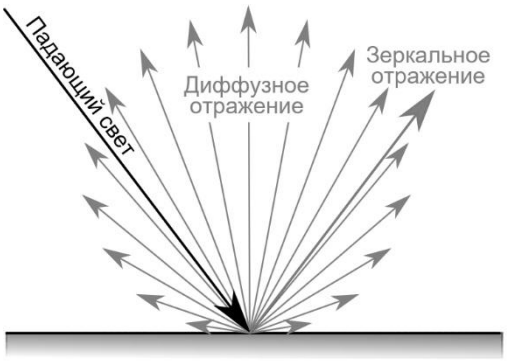


Рисунок 9Диффузионное отражение света

Интенсивность отражения света рассчитывается по формуле:

$$Id = I0Kd\cos\vartheta$$

I_0 – интенсивность излучения источника.

K_d – коэффициент, который учитывает свойства материала поверхности (от 0 до 1).

θ – угол между направлением на точечный источник света и нормалью к поверхности.

Диффузное отражение определяется как косинус угла между вектором нормали к поверхности и некоторым направлением, определяемым вектором S (рис. 3).

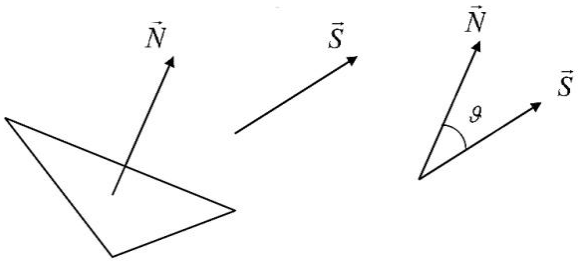


Рисунок 10

Источник света или наблюдатель находятся на бесконечности по отношению к некоторому элементу поверхности:

$$\vec{N} = \vec{N}(N_x, N_y, N_z)$$

вектор нормали к элементу поверхности :

$$\vec{S} = \vec{S}(S_x, S_y, S_z)$$

вектор определяющий некоторое направление в пространстве:

$$\cos\vartheta = \frac{\vec{S}\cdot\vec{N}}{\left|\vec{S}\right|\cdot\left|\vec{N}\right|} = \frac{S_xN_x + S_yN_y + S_zN_z}{\sqrt{S_x^2 + S_y^2 + S_z^2}\sqrt{N_x^2 + N_y^2 + N_z^2}}$$

Методы закрашивания Гуро и Фонга

Метод Гуро.

Предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями.

Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник.

Идея: закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней. Закрашивание граней по методу Гуро осуществляется в четыре этапа:

1. вычисляются нормали к каждой грани
2. определяются нормали в вершинах, нормаль в вершинах, нормаль в вершине определяется усреднением нормалей примыкающих граней

$$\vec{N}_a = \frac{\vec{N}_1 + \vec{N}_2 + \vec{N}_3}{3}$$

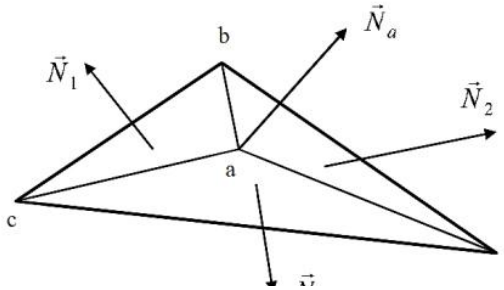


Рисунок 11

рис. 4 и определяется по формуле:

3. на основе нормалей в вершинах вычисляются значения интенсивностей в вершинах согласно выбранной модели отражения света.
4. закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Заполнение контура грани горизонталями в экранных координатах рис. 5.

$$I_1 = I_b + (I_c - I_b)\frac{Y - Y_b}{Y_c - Y_b}$$

$$I_2 = I_b + (I_a - I_b)\frac{Y - Y_b}{Y_a - Y_b}$$

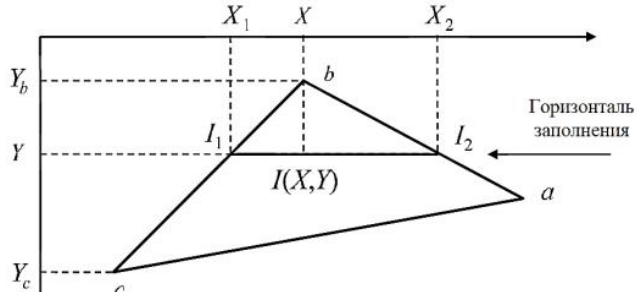


Рисунок 12

Метод Фонга.

Идея: используется интерполяция вектора нормали к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности.

Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями в вершинах.

Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности.

Если скорость распространения света в двух средах отличается, то на границе этих сред происходит преломление падающего светового луча. Преломленный луч лежит в той же плоскости, что и векторы V и п, а угол падения связан с углом преломления законом Снеллиуса.

$$\eta_1 \sin \Theta_i = \eta_2 \sin \Theta_t$$

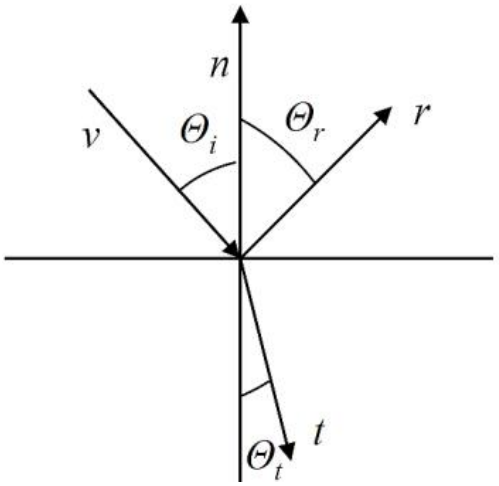


Рисунок 13 Вектор преломления луча

Ход работы:

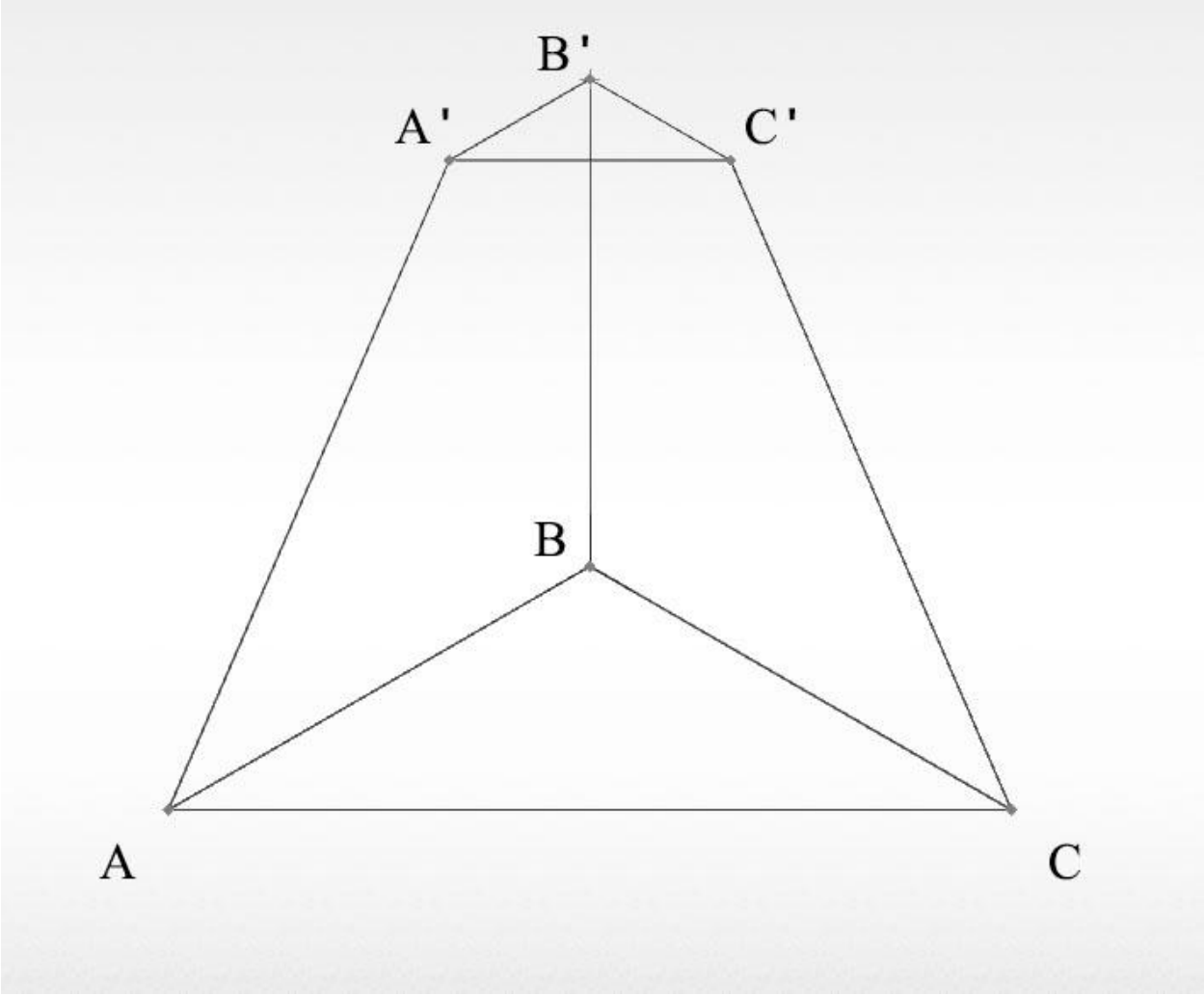
Сначала нам нужно построить изображение нашей усеченной пирамиды с удалением невидимых граней. А после этого спроецировать освещение.

Построение усеченной пирамиды с удалением НГ:

Первым делом нам потребуются координаты *вершин*, *камеры* и *область отображения в окне*. Координаты вершины можно задать следующим образом:

003001
300100
000333
111111

График:



Координаты камеры задаются в Сферической Системе Координат:

$r_v = 10$

ϕ_{vg}=20 (градусы)

θ_{vg}=110 (градусы)

Область отражения окна (пример):

$X_{LW} = 100$; $Y_{LW} = 200$
$X_{HW} = 800$; $Y_{HW} = 900$

Матрица координат у нас уже имеется. Построим матрицу пересчёта из мировой системы координат в видовую:

$$K_{view} := \begin{pmatrix} -\sin(\varphi_v) & \cos(\varphi_v) & 0 & 0 \\ -\cos(\theta_v)\cdot\cos(\varphi_v) & -\cos(\theta_v)\cdot\sin(\varphi_v) & \sin(\theta_v) & 0 \\ -\sin(\theta_v)\cdot\cos(\varphi_v) & -\sin(\theta_v)\cdot\sin(\varphi_v) & -\cos(\theta_v) & r_v \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

ϕ_V в данной матрице представляет собой ϕ_{Vg}, переведённую в радианы.
Вычисляем координаты вершин пирамиды в видовой системе координат. Для этого перемножаем матрицу пересчёта на матрицу координат – именно в такой последовательности, ибо так не нарушается правило перемножение матриц.

$$K := \begin{bmatrix} -\sin(0.35) & \cos(0.35) & 0 & 0 \\ -\cos(1.91)\cdot\cos(0.35) & -\cos(1.91)\cdot\sin(0.35) & \sin(1.91) & 0 \\ -\sin(1.91)\cdot\cos(0.35) & -\sin(1.91)\cdot\sin(0.35) & -\cos(1.91) & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.343 & 0.939 & 0 & 0 \\ 0.313 & 0.114 & 0.943 & 0 \\ -0.886 & -0.323 & 0.333 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$PIR := \begin{bmatrix} 0 & 0 & 3 & 0 & 0 & 1 \\ 3 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$PIR\_V := K \cdot PIR = \begin{bmatrix} 2.818 & 0 & -1.029 & 0.939 & 0 & -0.343 \\ 0.342 & 0 & 0.938 & 2.943 & 2.829 & 3.142 \\ 9.03 & 10 & 7.342 & 10.675 & 10.998 & 10.112 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Формируем матрицу из координат *проекций* вершин пирамиды на плоскость X_vY_v – плоскость в Видовой Системе Координат:

$$P := \begin{bmatrix} 2.818 & 0 & 1.029 & 0.939 & 0 & -0.343 \\ 0.342 & 0 & 0.938 & 2.943 & 2.829 & 3.142 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

В нашей матрице первая строка представляет собой вектор XV – координат проекций вершин на X_vY_v. А вторая строка вектор YV.
Далее определяем *габаритную область* проекции пирамиды на плоскость X_vY_v видовой СК – картинную плоскость. Для этого найдем координаты левого верхнего угла и правого нижнего.

$x_L := \min(XV)$ $y_H := \max(YV)$

$x_H := \max(XV)$ $y_L := \min(YV)$

X_L = -0.343; Y_H = 3.142;

X_H = 2.818; Y_L = 0;

Вычисляем параметры, необходимые для формирования *матрицы пересчёта* координат из плоскости X_vY_v видовой СК в оконную СК:

$\Delta x_W := x_{HW} - x_{LW}$ $\Delta x_W = 700$

$\Delta x := x_H - x_L$ $\Delta x = 3.161$

$\Delta y_W := y_{HW} - y_{LW}$ $\Delta y_W = 700$

$\Delta y := y_H - y_L$ $\Delta y = 3.142$

$k_x := \frac{\Delta x_W}{\Delta x}$ $k_x = 221.448$

$k_y := \frac{\Delta y_W}{\Delta v}$ $k_y = 222.788$

Матрица пересчёта:

$$T_{SW} := \begin{pmatrix} k_x & 0 & x_{LW} - k_x \cdot x_L \\ 0 & -k_y & y_{HW} + k_y \cdot y_L \\ 0 & 0 & 1 \end{pmatrix}$$

$$T := \begin{bmatrix} 221.448 & 0 & 100 - 221.448 \cdot (-0.343) \\ 0 & -222.788 & 900 + 222.788 \cdot 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 221.448 & 0 & 175.957 \\ 0 & -222.788 & 900 \\ 0 & 0 & 1 \end{bmatrix}$$

Вычисляем оконные координаты вершин пирамиды. Для этого умножим нашу сформированную матрицу пересчёта на матрицу координат X_vY_v Видовой СК. В полученную матрицу записываем числа, округлённые до целых:

$M1_W := T_{SW} \cdot P_{XY}$ $M_W := \overrightarrow{\text{round}(M1_W)}$ ***- Округление до целых***

$$\begin{pmatrix} x_a & x_b & x_c & x_d \\ y_a & y_b & y_c & y_d \\ q & q & q & q \end{pmatrix} := M_W$$

$$M1 := T \cdot P = \begin{bmatrix} 799.997 & 175.957 & 403.827 & 383.896 & 175.957 & 100 \\ 823.807 & 900 & 691.025 & 244.335 & 269.733 & 200 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$M := \overrightarrow{\text{round}(M1, 0)} = \begin{bmatrix} 800 & 176 & 404 & 384 & 176 & 100 \\ 824 & 900 & 691 & 244 & 270 & 200 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Далее нужно определить видимость граней.

Для этого первым делом посчитаем декартовы координаты камеры – вектор, определяющий положение камеры. Формула:

$$R_C := \begin{pmatrix} r_v \cdot \sin(\theta_v) \cdot \cos(\varphi_v) \\ r_v \cdot \sin(\theta_v) \cdot \sin(\varphi_v) \\ r_v \cdot \cos(\theta_v) \end{pmatrix}$$

Высчитываем:

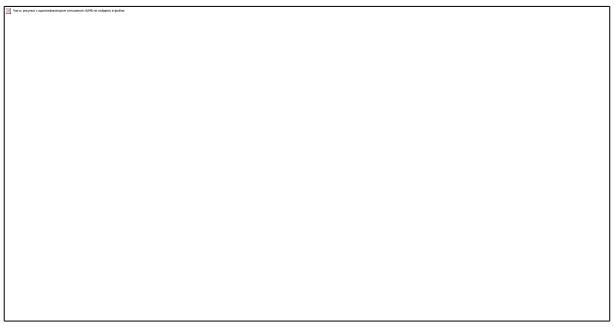
$$r := 10 \qquad O := 1.92 \qquad f := 0.35$$

$$R := \begin{bmatrix} r \cdot \sin(O) \cdot \cos(f) \\ r \cdot \sin(O) \cdot \sin(f) \\ r \cdot \cos(O) \end{bmatrix} = \begin{bmatrix} 8.827 \\ 3.222 \\ -3.421 \end{bmatrix}$$

Далее высчитываем декартовы координаты вектора внешней нормали к каждой грани пирамиды. Для этого следует найти координаты векторов, составляющие каждую из граней, например:

После этого, применив векторное произведение найти вектор внешней нормали:

Порядок векторов определяется правосторонним движением. То есть для грани ABB’A’:



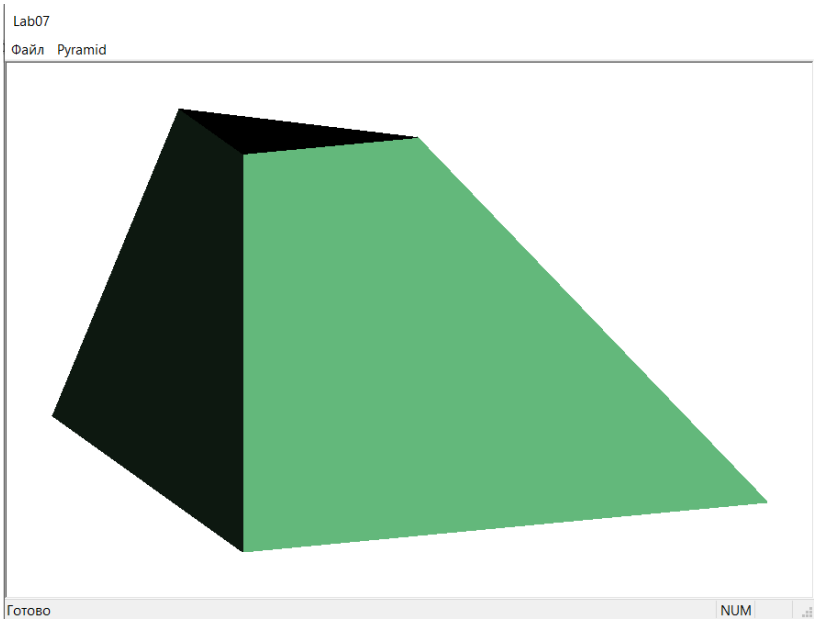
Видимость каждой из грани определяем нахождением косинуса угла между вектором внешней нормали и вектором камеры. Для этого нужно произведение векторов камеры и внешней нормали разделить на произведение их модулей (скалярное произведение):

Так как у нас шестиугольная пирамида, то у нас имеется два основания. Определяем их видимость так: если верхнее основание видимо, то нижнее нет.

Заполнение освещения:
Для закрашивания используется один из методов, минимально раскрытых в теоретических сведениях – методы Гуро и Фонга. Однако они просто заполняют грани нужным цветом.

Для создания нужного оттенка, следует умножить раннее полученный косинус угла на каждый из цветов RGB.

В итоге, должны получить что-то вроде этого (как пример):



Реализация приложения на C++:

Для отображения графики используется класс **ChildView**, использующий библиотеку MFC. Листинг **ChildView.h**:

```
#pragma once
class CChildView : public CWnd
{
// Создание - конструктор.
public:
	// Объект пирамиды.
	CPyramid PIR;
	// Область в окне, в котором происходит рисование пирамиды.
	CRect WinRect;
	// Точка наблюдения в сферической СК. Задается матрицей.
	CMatrix PView;
	int Index;

	CChildView();
	virtual ~CChildView();

// Действия при выборе пункта меню
	afx_msg void OnPaint();
	afx_msg void OnPyramid();
	afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
	afx_msg void OnSize(UINT nType, int cx, int cy);

// Переопределение
protected:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
// Созданные функции схемы сообщений
	DECLARE_MESSAGE_MAP()
};
```

Данный класс содержит три логически разделенных группы свойств:

1. Поля:
    1. **PIR** – объект, представляющий пирамиду.
    2. **WinRect** – объект, представляющий окно, в котором пирамида рисуется.
    3. **PView** – объект, представляющий собой точку наблюдения, записанный матрицей.
    4. **Index** – отображает текущий индекс поведения.
2. Необходимая реализация MFC, а именно – карта сообщений. (См. приложение)
3. Методы с действиями пользователя:
    1. **OnPaint()** – Вызывается при создании окна, чтобы рисовать нужные объекты.
    2. **OnPyramid()** – Вызывается при выборе в меню Pyramid→ColorDraw.
    3. **OnKeyDown()** – Вызывается при нажатии клавиш в приложении.
    4. **OnSize()** – Вызывается при любом изменении окна приложения.

Реализация CPyramid, CMatrix, CRect (LibGraph):

Сначала реализуем класс **CMatrix**, представляющий собой матрицу, или вектор. Его (класса) методы являются арифметическими действиями над матрицами. Он имеет структуру, приведённую в листинге ниже.

```
#pragma once
#include <fstream>
using namespace std;

class CMatrix
{
	// Поля.
	double **array;
	int n_rows = 1; // Число строк
	int n_cols = 1; // Число столбцов
public:
	// Конструкторы/Деструкторы.
	CMatrix();			// Конструктор по умолчанию (1 на 1)
	CMatrix(int, int);		// Конструктор
	CMatrix(int);	// Конструктор вектора (один столбец)
	CMatrix(const CMatrix&);	// Конструктор копирования
	~CMatrix();

		// Перегрузка операторов.
		// Выбор элемента матрицы по индексу.
		double& operator()(int, int);
		// Выбор элемента вектора по индексу.
		double& operator()(int);
		// Оператор "Присвоить":	M1=M2.
		CMatrix operator=(const CMatrix&);
		// Оператор "Произведение": M1*M2.
		CMatrix operator*(CMatrix&);
		// Оператор "+": M1+M2.
		CMatrix operator+(CMatrix&);
		// Оператор "-": M1-M2.
		CMatrix operator-(CMatrix&);

		// Методы.
		// Возвращает число строк.
		int rows()const { return n_rows; };
		// Возвращает число столбцов.
		int cols()const { return n_cols; };
		// Возвращает матрицу, транспонированную к текущей.
		CMatrix Transp();
		// Возвращает строку по номеру.
		CMatrix GetRow(int);
		CMatrix GetRow(int, int, int);
		// Возвращает столбец по номеру.
		CMatrix GetCol(int);
		CMatrix GetCol(int, int, int);
		// Изменяет размер матрицы с уничтожением данных.
		CMatrix RedimMatrix(int, int);
```

```
            // Изменяет размер матрицы с сохранением данных, которые
можно сохранить.
            CMatrix RedimData(int, int);
            // Изменяет размер матрицы с уничтожением данных.
            CMatrix RedimMatrix(int);
            // Изменяет размер матрицы с сохранением данных, которые
можно сохранить.
            CMatrix RedimData(int);
            // Максимальный элемент матрицы
            double MaxElement();
            // Минимальный элемент матрицы
            double MinElement();
};
```

Его реализация: (Осторожно, много букаф…)

```
#include "stdafx.h"
#include "CMatrix.h"

//---------------------------------------------------------------------
-------------
// Конструкторы/Деструкторы.
//---------------------------------------------------------------------
-------------

CMatrix::CMatrix()
{
     //   n_rows = 1;
     //   n_cols = 1;

     array = new double*[n_rows];
     for (int i = 0; i < n_rows; i++)
          array[i] = new double[n_cols];

     for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++)
               array[i][j] = 0;
}

CMatrix::CMatrix(int Nrow, int Ncol)
// Nrow - число строк
// Ncol - число столбцов
{
     n_rows = Nrow;
     n_cols = Ncol;

     array = new double*[n_rows];
     for (int i = 0; i < n_rows; i++)
          array[i] = new double[n_cols];

     for (int i = 0; i < n_rows; i++)
```

```
            for (int j = 0; j < n_cols; j++)
                  array[i][j] = 0;
}

// Создание вектора.
CMatrix::CMatrix(int Nrow)
// Nrow - число строк
{
      n_rows = Nrow;
      // n_cols = 1;

      array = new double*[n_rows];
      for (int i = 0; i < n_rows; i++)
            array[i] = new double[n_cols];

      for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++)
                  array[i][j] = 0;
}

CMatrix::CMatrix(const CMatrix& M)
// Конструктор копирования
{
      n_rows = M.n_rows;
      n_cols = M.n_cols;

      array = new double* [n_rows];
      for (int i = 0; i < n_rows; i++)
            array[i] = new double[n_cols];

      for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++)
                  array[i][j] = M.array[i][j];
}

CMatrix::~CMatrix()
{
      for (int i = 0; i < n_rows; i++)
            delete array[i];

      delete array;
}

//-------------------------------------------------------------------
-------------
// Перегрузка операторов.
//-------------------------------------------------------------------
-------------

double &CMatrix::operator()(int i, int j)
```

```
// i - номер строки
// j - номер столбца
{
     if ((i > n_rows - 1) || (j > n_cols - 1))     //  проверка выхода
за диапазон индексации массива.
     {
          TCHAR* error = _T("CMatrix::operator(int,int): выход индекса
за границу диапазона ");
          MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
          exit(1);
     }

     return array[i][j];
}

double &CMatrix::operator()(int i)
// i - номер строки для вектора
{
     if (n_cols > 1)     //  Число столбцов больше одного
     {
          char* error = "CMatrix::operator(int): объект не вектор -
число столбцов больше 1 ";
          MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
          exit(1);
     }

     if (i > n_rows - 1)     //  проверка выхода за диапазон
индексации массива.
     {
          TCHAR* error = TEXT("CMatrix::operator(int): выход индекса
за границу диапазона ");
          MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
          exit(1);
     }

     return array[i][0];
}

CMatrix CMatrix::operator=(const CMatrix& M)
// Оператор присваивания M1=M
{
     if (this == &M)
          return *this;

     int nn = M.rows();
     int mm = M.cols();

     if ((n_rows == nn) && (n_cols == mm))
     {
          for (int i = 0; i < n_rows; i++)
```

```
                for (int j = 0; j < n_cols; j++)
                     array[i][j] = M.array[i][j];
     }
     else
     {
          TCHAR* error = TEXT("CMatrix::operator=: несоответствие
размерностей матриц");
          MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
          exit(1);
     }

     return *this;
}

CMatrix CMatrix::operator+(CMatrix& M)
// Оператор M1 + M2
{
     int bb = (n_rows == M.rows()) && (n_cols == M.cols());
     if (!bb)
     {
          char* error = "CMatrix::operator(+): несоответствие
размерностей матриц ";
          MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
          exit(1);
     }

     CMatrix Temp(*this);
     for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++)
               Temp(i, j) += M(i, j);
     return Temp;
}

CMatrix CMatrix::operator-(CMatrix& M)
// Оператор M1-M2
{
     int bb = (n_rows == M.rows()) && (n_cols == M.cols());
     if (!bb)
     {
          char* error = "CMatrix::operator(-): несоответствие
размерностей матриц ";
          MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
          exit(1);
     }

     CMatrix Temp(*this);
     for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++)
               Temp(i, j) -= M(i, j);
     return Temp;
```

```
}

CMatrix CMatrix::operator*(CMatrix& M)
// Умножение на матрицу M
{
      double sum;
      int nn = M.rows();
      int mm = M.cols();
      CMatrix Temp(n_rows, mm);

      if (n_cols == nn)
      {
            for (int i = 0; i < n_rows; i++)
            {
                  for (int j = 0; j < mm; j++)
                  {
                        sum = 0;
                        for (int k = 0; k < n_cols; k++)
                              sum += (*this)(i, k) * M(k, j);
                        Temp(i, j) = sum;
                  }
            }
      }
      else
      {
            TCHAR* error = TEXT("CMatrix::operator*: несоответствие
размерностей матриц ");
            MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
            exit(1);
      }

      return Temp;
}

//-------------------------------------------------------------------
-------------
// Методы.
//-------------------------------------------------------------------
-------------

CMatrix CMatrix::Transp()
// Возвращает матрицу,транспонированную к (*this)
{
      CMatrix Temp(n_cols, n_rows);

      for (int i = 0; i < n_cols; i++)
            for (int j = 0; j < n_rows; j++)
                  Temp(i, j) = array[j][i];

      return Temp;
```

```
}

CMatrix CMatrix::GetRow(int k)
// Возвращает строку матрицы по номеру k
{
	if (k > n_rows - 1)
	{
		char* error = "CMatrix::GetRow(int k): параметр k превышает
число строк ";
		MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
		exit(1);
	}

	CMatrix M(1, n_cols);
	for (int i = 0; i < n_cols; i++)
		M(0, i) = (*this)(k, i);

	return M;
}
//-------------------------------------------------------------------
-------------
CMatrix CMatrix::GetRow(int k, int n, int m)
// Возвращает подстроку из строки матрицы с номером k
// n - номер первого элемента в строке
// m - номер последнего элемента в строке
{
	int b1 = (k >= 0) && (k < n_rows);
	int b2 = (n >= 0) && (n <= m);
	int b3 = (m >= 0) && (m < n_cols);
	int b4 = b1 && b2&&b3;
	if (!b4)
	{
		char* error = "CMatrix::GetRow(int k,int n, int m):ошибка в
параметрах ";
		MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
		exit(1);
	}

	int nCols = m - n + 1;
	CMatrix M(1, nCols);
	for (int i = n; i <= m; i++)
		M(0, i - n) = (*this)(k, i);

	return M;
}

CMatrix CMatrix::GetCol(int k)
// Возвращает столбец матрицы по номеру k
{
	if (k > n_cols - 1)
```

```
	{
		char* error = "CMatrix::GetCol(int k): параметр k превышает
число столбцов ";
		MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
		exit(1);
	}

	CMatrix M(n_rows, 1);
	for (int i = 0; i < n_rows; i++)
		M(i, 0) = (*this)(i, k);

	return M;
}

CMatrix CMatrix::GetCol(int k, int n, int m)
// Возвращает подстолбец из столбца матрицы с номером k
// n - номер первого элемента в столбце
// m - номер последнего элемента в столбце
{
	int b1 = (k >= 0) && (k < n_cols);
	int b2 = (n >= 0) && (n <= m);
	int b3 = (m >= 0) && (m < n_rows);
	int b4 = b1 && b2&&b3;
	if (!b4)
	{
		char* error = "CMatrix::GetCol(int k,int n, int m):ошибка в
параметрах ";
		MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
		exit(1);
	}

	int nRows = m - n + 1;
	CMatrix M(nRows, 1);
	for (int i = n; i <= m; i++)
		M(i - n, 0) = (*this)(i, k);

	return M;
}

CMatrix CMatrix::RedimMatrix(int NewRow, int NewCol)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol - новое число столбцов
{
	for (int i = 0; i < n_rows; i++)
		delete array[i];
	delete array;

	n_rows = NewRow;
	n_cols = NewCol;
```

```
	array = new double*[n_rows];
	for (int i = 0; i < n_rows; i++)
		array[i] = new double[n_cols];

	for (int i = 0; i < n_rows; i++)
		for (int j = 0; j < n_cols; j++)
			array[i][j] = 0;

	return (*this);
}

CMatrix CMatrix::RedimData(int NewRow, int NewCol)
// Изменяет размер матрицы с сохранением данных, которые можно
сохранить
// NewRow - новое число строк
// NewCol - новое число столбцов
{
	CMatrix Temp = (*this);
	this->RedimMatrix(NewRow, NewCol);

	int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() :
(*this).rows();
	int min_cols = Temp.cols() < (*this).cols() ? Temp.cols() :
(*this).cols();

	for (int i = 0; i < min_rows; i++)
		for (int j = 0; j < min_cols; j++)
			(*this)(i, j) = Temp(i, j);

	return (*this);
}

CMatrix CMatrix::RedimMatrix(int NewRow)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol=1
{
	for (int i = 0; i < n_rows; i++)
		delete array[i];
	delete array;

	n_rows = NewRow;
	n_cols = 1;

	array = new double*[n_rows];
	for (int i = 0; i < n_rows; i++)
		array[i] = new double[n_cols];

	for (int i = 0; i < n_rows; i++)
```

```
            for (int j = 0; j < n_cols; j++)
                  array[i][j] = 0;

      return (*this);
}

CMatrix CMatrix::RedimData(int NewRow)
// Изменяет размер матрицы с сохранением данных, которые можно
сохранить
// NewRow - новое число строк
// NewCol = 1
{
      CMatrix Temp = (*this);
      this->RedimMatrix(NewRow);

      int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() :
(*this).rows();

      for (int i = 0; i < min_rows; i++)
            (*this)(i) = Temp(i);
      return (*this);
}

double CMatrix::MaxElement()
// Максимальное значение элементов матрицы
{
      double max = (*this)(0, 0);
      for (int i = 0; i < (this->rows()); i++)
            for (int j = 0; j < (this->cols()); j++)
                  if ((*this)(i, j) > max)
                        max = (*this)(i, j);
      return max;
}

double CMatrix::MinElement()
// Минимальное значение элементов матрицы
{
      double min = (*this)(0, 0);
      for (int i = 0; i < (this->rows()); i++)
            for (int j = 0; j < (this->cols()); j++)
                  if ((*this)(i, j) < min)
                        min = (*this)(i, j);
      return min;
}
```

Класс **CPyramid**, представляющий собой пирамиду имеет такую структуру:

```
class CPyramid
{
      CMatrix Vertices;          // Координаты вершин
      void GetRect(CMatrix& Vert, CRectD& RectView);
```

```
public:
	CPyramid();
	void ColorDraw(CDC& dc, CMatrix& PView, CRect& RW, COLORREF
Color);
};
```

В нём мы имеем поле **CMatrix** *Vertices*, представляющее собой координаты вершины пирамиды. Далее – конструктор и два метода. Первый, закрытый, метод **GetRect()** используется во втором, открытом, методе **ColorDraw()**, который требуется реализовать по заданию. Метод **ColorDraw()** же будет вызываться позже в **ChildView**.

Реализация класса **CPyramid**:

```
#include "stdafx.h"

CPyramid::CPyramid()
{
	Vertices.RedimMatrix(4, 6);
	// Координаты вершин - столбцы
	Vertices(0, 2) = 3;
	Vertices(0, 5) = 1;
	Vertices(1, 0) = 3;
	Vertices(1, 3) = 1;
	Vertices(2, 3) = 3;
	Vertices(2, 4) = 3;
	Vertices(2, 5) = 3;
	for (int i = 0; i < 6; i++)
		Vertices(3, i) = 1;
}

void CPyramid::ColorDraw(CDC& dc, CMatrix& PView, CRect& RW, COLORREF
Color)
{
	BYTE red = GetRValue(Color);
	BYTE green = GetGValue(Color);
	BYTE blue = GetBValue(Color);

	// Преобразуем сферические к-ты в декартовы.
	CMatrix ViewCart = SphereToCart(PView);

	// Матрица пересчета из МСК в ВСК.
	CMatrix MV = CreateViewCoord(PView(0), PView(1), PView(2));
	// Получаем матрицу к-т пирамиды в ВСК
	CMatrix ViewVert = MV * Vertices;

	CRectD RectView;
	// Получаем прямоугольник, охватывающий пирамиду.
	GetRect(ViewVert, RectView);
	// М-ца пересчета из ВСК в ОСК.
	CMatrix MW = SpaceToWindow(RectView, RW);
```

```
	CPoint MasVert[6];
	CMatrix V(3);
	V(2) = 1;
	// Пересчитываем к-ты в ОСК и записываем в массивы MasVert.
	for (int i = 0; i < 6; i++)
	{
		V(0) = ViewVert(0, i); // x
		V(1) = ViewVert(1, i); // y
		V = MW * V;
		MasVert[i].x = (int)V(0);
		MasVert[i].y = (int)V(1);
	}

	CMatrix R1(3), R2(3), VN(3);
	// Коэффициент преломления.
	double sm;
	for (int i = 0; i < 3; i++)
	{
		CMatrix VE = Vertices.GetCol(i+3, 0, 2);   // получаем к-ты
точки верхнего основания
		int k;
		if (i == 2) k = 0;
		else k = i + 1;

		R1 = Vertices.GetCol(i, 0, 2);        // текущая точка
основания
		R2 = Vertices.GetCol(k, 0, 2);        // следующая точка
основания

		CMatrix V1 = R2 - R1;                 // вектор основания

		CMatrix V2 = VE - R1;                 // вектор между
основанием и вершиной
		VN = VectorMult(V2, V1);              // векторное
произведение - вектор внешней нормали к грани

		sm = CosV1V2(VN, ViewCart);                 // косинус между
вектором нормали к грани и вектором точки наблюдения

		if (sm >= 0)                                // грань видима
(острый угол) - рисуем боковую грань
		{
			CPen Pen(PS_SOLID, 2,           // задаем перо
				RGB(sm*sm*red, sm*sm*green, sm*sm*blue));  //
берем байты цветов и уменьшаем их цвет квадратом косинуса
			CPen* pOldPen = dc.SelectObject(&Pen);     // передаем
перо в контекст рисования

			CBrush Brus(RGB(sm*sm*red, sm*sm*green, sm*sm*blue));
	// задаем изменение цвета при смене угла
```

```
                CBrush* pOldBrush = dc.SelectObject(&Brus);
            // передаем цвет в контекст памяти
                CPoint MasVertR[4] = {
MasVert[i],MasVert[k],MasVert[k+3],MasVert[i+3] };// Стороны
треугольника
                dc.Polygon(MasVertR, 4);        // боковая грань

                dc.SelectObject(pOldBrush);          // освобождаем
контексты памяти
                dc.SelectObject(pOldPen);
            }
      }

      VN = VectorMult(R1, R2);
      sm = CosV1V2(VN, ViewCart);
      if (sm >= 0)
      {
            CBrush Brus(RGB(sm*0.3, sm*0.3, sm*0.3));
            CBrush* pOldBrush = dc.SelectObject(&Brus);
            dc.Polygon(MasVert, 3);    // Основание
            dc.SelectObject(pOldBrush);
      }
      else
      {
            CBrush Brus(RGB(sm*0.7, sm*0.7, sm*0.7));
            CBrush* pOldBrush = dc.SelectObject(&Brus);
            dc.Polygon(MasVert+3, 3);  // верхнее основание
            dc.SelectObject(pOldBrush);
      }

}

void CPyramid::GetRect(CMatrix& Vert, CRectD& RectView)
// Вычисляет координаты прямоугольника,охватывающего проекцию
// пирамиды на плоскость XY в ВИДОВОЙ системе координат
// Ver - координаты вершин (в столбцах)
// RectView - проекция - охватывающий прямоугольник
{
      CMatrix V = Vert.GetRow(0);           // x - координаты
      double xMin = V.MinElement();
      double xMax = V.MaxElement();

      V = Vert.GetRow(1);                   // y - координаты
      double yMin = V.MinElement();
      double yMax = V.MaxElement();
      RectView.SetRectD(xMin, yMax, xMax, yMin);
}
```

Класс **LibGraph** содержит в себе структуру CRectD – прямоугольник, а также набор функций, необходимых для расчётов:

```
#ifndef LIBGRAPH
```

```
#define LIBGRAPH 1
const double pi = 3.14159;
typedef double(*pfunc2)(double, double);   // Указатель на функцию

struct CSizeD
{
      double cx;
      double cy;
};
//---------------------------------------------------------------------
---------------
struct CRectD
{
      double left;
      double top;
      double right;
      double bottom;
      CRectD() { left = top = right = bottom = 0; };
      CRectD(double l, double t, double r, double b);
      void SetRectD(double l, double t, double r, double b);
      CSizeD SizeD();
};

//---------------------------------------------------------------------
----------------

CMatrix SpaceToWindow(CRectD& rs, CRect& rw);
// Возвращает матрицу пересчета координат из мировых в оконные
// rs - область в мировых координатах - double
// rw - область в оконных координатах - int
//---------------------------------------------------------------------
---------------
void SetMyMode(CDC& dc, CRect& RS, CRect& RW);  //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS -  область в мировых координатах - int
// RW -     Область в оконных координатах - int
//---------------------------------------------------------------------
-------------
CMatrix CreateTranslate2D(double dx, double dy);
// Формирует матрицу для преобразования координат объекта при его
смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении
начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном
положении объекта
//---------------------------------------------------------------------
--------------
```

```
CMatrix CreateTranslate3D(double dx, double dy, double dz);
// Формирует матрицу для преобразования координат объекта при его
смещении
// на dx по оси X, на dy по оси Y,на dz по оси Z в фиксированной
системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении
начала
// системы координат на -dx оси X,на -dy по оси Y, на -dz по оси Z
// при фиксированном положении объекта
//-----------------------------------------------------------------
-------------
CMatrix CreateRotate2D(double fi);
// Формирует матрицу для преобразования координат объекта при его
повороте
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//-----------------------------------------------------------------
------------
CMatrix CreateRotate3DZ(double fi);
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси Z
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат вокруг оси Z на угол -fi при фиксированном
положении объекта
// fi - угол в градусах

//-----------------------------------------------------------------
------------
CMatrix CreateRotate3DX(double fi);
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси X
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат вокруг оси X на угол -fi при фиксированном
положении объекта
// fi - угол в градусах
```

```
//-------------------------------------------------------------------
------------
CMatrix CreateRotate3DY(double fi);
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси Y
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат вокруг оси Y на угол -fi при фиксированном
положении объекта
// fi - угол в градусах

//-------------------------------------------------------------------
------------
CMatrix CreateViewCoord(double r, double fi, double q);
// Создает матрицу пересчета точки из мировой системы координат в
видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы
координат)
// в мировой сферической системе координат( углы fi и q в градусах)
//-------------------------------------------------------------------
-----------
CMatrix VectorMult(CMatrix& V1, CMatrix& V2);
// Вычисляет векторное произведение векторов V1 и V2
//-------------------------------------------------------------------
-----------
double ScalarMult(CMatrix& V1, CMatrix& V2);
// Вычисляет скалярное произведение векторов V1 и V2
//-------------------------------------------------------------------
-----------
double ModVec(CMatrix& V);
// Вычисляет модуль вектора V
//-------------------------------------------------------------------
----------
double CosV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет КОСИНУС угла между векторами V1 и V2
//-------------------------------------------------------------------
---------
double AngleV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет угол между векторами V1 и V2 в градусах
//-------------------------------------------------------------------
--------------
CMatrix SphereToCart(CMatrix& PView);
// Преобразует сферические координаты PView  точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчетот оси Z), град.
```

```
// Результат: R(0)- x, R(1)- y, R(2)- z
//---------------------------------------------------------------------
---------------
void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD&
PR);
// Вычисляет координаты проекции охватывающего фигуру паралелепипеда
на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая
поверхность
// PView - координаты точки наблюдения в мировой сферической системе
координат
// PR - проекция
//---------------------------------------------------------------------
---------------

#endif
```

Его реализация:

```
#include "stdafx.h"

CRectD::CRectD(double l, double t, double r, double b)
{
	left = l;
	top = t;
	right = r;
	bottom = b;
}
//---------------------------------------------------------------------
----------
void CRectD::SetRectD(double l, double t, double r, double b)
{
	left = l;
	top = t;
	right = r;
	bottom = b;
}

//---------------------------------------------------------------------
----------
CSizeD CRectD::SizeD()
{
	CSizeD cz;
	cz.cx = fabs(right - left);	// Ширина прямоугольной области
	cz.cy = fabs(top - bottom);	// Высота прямоугольной области
	return cz;
}

//---------------------------------------------------------------------
```

```
----------
CMatrix SpaceToWindow(CRectD& RS, CRect& RW)
// Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
{
     CMatrix M(3, 3);
     CSize sz = RW.Size();  // Размер области в ОКНЕ
     int dwx = sz.cx;     // Ширина
     int dwy = sz.cy;     // Высота
     CSizeD szd = RS.SizeD(); // Размер области в МИРОВЫХ координатах

     double dsx = szd.cx;    // Ширина в мировых координатах
     double dsy = szd.cy;    // Высота в мировых координатах

     double kx = (double)dwx / dsx;   // Масштаб по x
     double ky = (double)dwy / dsy;   // Масштаб по y


     M(0, 0) = kx;  M(0, 1) = 0;    M(0, 2) = (double)RW.left - kx *
RS.left;
     M(1, 0) = 0;   M(1, 1) = -ky;  M(1, 2) = (double)RW.bottom + ky *
RS.bottom;
     M(2, 0) = 0;   M(2, 1) = 0;    M(2, 2) = 1;
     return M;
}



 //-----------------------------------------------------------------
-----------

void SetMyMode(CDC& dc, CRect& RS, CRect& RW)  //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS -  область в мировых координатах - int
// RW -     Область в оконных координатах - int
{
     int dsx = RS.right - RS.left;
     int dsy = RS.top - RS.bottom;
     int xsL = RS.left;
     int ysL = RS.bottom;

     int dwx = RW.right - RW.left;
     int dwy = RW.bottom - RW.top;
     int xwL = RW.left;
     int ywH = RW.bottom;

     dc.SetMapMode(MM_ANISOTROPIC);
     dc.SetWindowExt(dsx, dsy);
```

```
      dc.SetViewportExt(dwx, -dwy);
      dc.SetWindowOrg(xsL, ysL);
      dc.SetViewportOrg(xwL, ywH);
}
//--------------------------------------------------------------------
----------------
CMatrix CreateTranslate2D(double dx, double dy)
// Формирует матрицу для преобразования координат объекта при его
смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении
начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном
положении объекта
{
      CMatrix TM(3, 3);
      TM(0, 0) = 1; TM(0, 2) = dx;
      TM(1, 1) = 1;  TM(1, 2) = dy;
      TM(2, 2) = 1;
      return TM;
}

//--------------------------------------------------------------------
-----------------
CMatrix CreateTranslate3D(double dx, double dy, double dz)
// Формирует матрицу для преобразования координат объекта при его
смещении
// на dx по оси X, на dy по оси Y,на dz по оси Z в фиксированной
системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении
начала
// системы координат на -dx оси X,на -dy по оси Y, на -dz по оси Z
// при фиксированном положении объекта
{
      CMatrix TM(4, 4);
      for (int i = 0; i < 4; i++) TM(i, i) = 1;
      TM(0, 3) = dx;
      TM(1, 3) = dy;
      TM(2, 3) = dz;
      return TM;
}



//--------------------------------------------------------------------
----------------
CMatrix CreateRotate2D(double fi)
// Формирует матрицу для преобразования координат объекта при его
```

```
повороте
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
      double fg = fmod(fi, 360.0);
      double ff = (fg / 180.0)*pi; // Перевод в радианы
      CMatrix RM(3, 3);
      RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
      RM(1, 0) = sin(ff);  RM(1, 1) = cos(ff);
      RM(2, 2) = 1;
      return RM;
}
//-------------------------------------------------------------------
------------

CMatrix CreateRotate3DZ(double fi)
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси Z
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат вокруг оси Z на угол -fi при фиксированном
положении объекта
// fi - угол в градусах
{
      double fg = fmod(fi, 360.0);
      double ff = (fg / 180.0)*pi; // Перевод в радианы
      CMatrix RM(4, 4);
      RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
      RM(1, 0) = sin(ff);  RM(1, 1) = cos(ff);
      RM(2, 2) = 1;
      RM(3, 3) = 1;
      return RM;
}
//-------------------------------------------------------------------
------------
CMatrix CreateRotate3DX(double fi)
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси X
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
```

```
начала
// системы координат вокруг оси X на угол -fi при фиксированном
положении объекта
// fi - угол в градусах
{
      double fg = fmod(fi, 360.0);
      double ff = (fg / 180.0)*pi; // Перевод в радианы
      CMatrix RM(4, 4);
      RM(0, 0) = 1;
      RM(1, 1) = cos(ff); RM(1, 2) = -sin(ff);
      RM(2, 1) = sin(ff); RM(2, 2) = cos(ff);
      RM(3, 3) = 1;
      return RM;
}
//-----------------------------------------------------------------
------------
CMatrix CreateRotate3DY(double fi)
// Формирует матрицу для преобразования координат объекта при его
повороте вокруг оси Y
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе
координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте
начала
// системы координат вокруг оси Y на угол -fi при фиксированном
положении объекта
// fi - угол в градусах
{
      double fg = fmod(fi, 360.0);
      double ff = (fg / 180.0)*pi; // Перевод в радианы
      CMatrix RM(4, 4);
      RM(0, 0) = cos(ff); RM(0, 2) = sin(ff);
      RM(1, 1) = 1;
      RM(2, 0) = -sin(ff); RM(2, 2) = cos(ff);
      RM(3, 3) = 1;
      return RM;
}

//-----------------------------------------------------------------
------------
CMatrix VectorMult(CMatrix& V1, CMatrix& V2)
// Вычисляет векторное произведение векторов V1 и V2
{

      int b1 = (V1.cols() == 1) && (V1.rows() == 3);
      int b2 = (V2.cols() == 1) && (V2.rows() == 3);
      int b = b1 && b2;
      if (!b)
      {
            //char* error="VectorMult: неправильные размерности
```

```
векторов! ";
          const TCHAR error[] = _T("VectorMult: неправильные
размерности векторов! ");
          MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
          exit(1);
     }
     CMatrix W(3);
     W(0) = V1(1)*V2(2) - V1(2)*V2(1);
     //double x=W(0);
     W(1) = -(V1(0)*V2(2) - V1(2)*V2(0));
     //double y=W(1);
     W(2) = V1(0)*V2(1) - V1(1)*V2(0);
     //double z=W(2);
     return W;
}

//-----------------------------------------------------------------
-----------
double ScalarMult(CMatrix& V1, CMatrix& V2)
// Вычисляет скалярное произведение векторов V1 и V2
{
     int b1 = (V1.cols() == 1) && (V1.rows() == 3);
     int b2 = (V2.cols() == 1) && (V2.rows() == 3);
     int b = b1 && b2;
     if (!b)
     {
          char* error = "ScalarMult: неправильные размерности
векторов! ";
          //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
          exit(1);
     }
     double p = V1(0)*V2(0) + V1(1)*V2(1) + V1(2)*V2(2);
     return p;
}
//-----------------------------------------------------------------
-----
double ModVec(CMatrix& V)
// Вычисляет модуль вектора V
{
     int b = (V.cols() == 1) && (V.rows() == 3);
     if (!b)
     {
          char* error = "ModVec: неправильнfz размерность вектора! ";
          //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
          exit(1);
     }
     double q = sqrt(V(0)*V(0) + V(1)*V(1) + V(2)*V(2));
     return q;
}
//-----------------------------------------------------------------
```

```
-----
double CosV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет КОСИНУС угла между векторами V1 и V2
{
      double modV1 = ModVec(V1);
      double modV2 = ModVec(V2);
      int b = (modV1 < 1e-7) || (modV2 < 1e-7);
      if (b)
      {
            char* error = "CosV1V2: модуль одного или обоих векторов <
1e-7!";
            //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
            exit(1);
      }
      int b1 = (V1.cols() == 1) && (V1.rows() == 3);
      int b2 = (V2.cols() == 1) && (V2.rows() == 3);
      b = b1 && b2;
      if (!b)
      {
            char* error = "CosV1V2: неправильные размерности векторов!
";
            //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
            exit(1);
      }
      double cos_f = ScalarMult(V1, V2) / (modV1*modV2);
      return cos_f;
}


//-------------------------------------------------------------------
-----
double AngleV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет угол между векторами V1 и V2 в градусах
{
      double modV1 = ModVec(V1);
      double modV2 = ModVec(V2);
      int b = (modV1 < 1e-7) || (modV2 < 1e-7);
      if (!b)
      {
            char* error = "AngleV1V2: модуль одного или обоих векторов <
1e-7!";
            //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
            exit(1);
      }
      int b1 = (V1.cols() == 1) && (V1.rows() == 3);
      int b2 = (V2.cols() == 1) && (V2.rows() == 3);
      b = b1 && b2;
      if (!b)
      {
            char* error = "AngleV1V2: неправильные размерности векторов!
```

```
";
            //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
            exit(1);
      }
      double cos_f = ScalarMult(V1, V2) / (modV1*modV2);
      if (fabs(cos_f) > 1)
      {
            char* error = "AngleV1V2: модуль cos(f)>1! ";
            //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
            exit(1);
      }
      double f;
      if (cos_f > 0)f = acos(cos_f);
      else f = pi - acos(cos_f);
      double fg = (f / pi) * 180;
      return fg;
}
//---------------------------------------------------------------------
------------
CMatrix CreateViewCoord(double r, double fi, double q)
// Создает матрицу пересчета (4x4) точки из мировой системы координат
в видо     вую.
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ (начало видовой системы
координат)
// в мировой сферической системе координат (углы fi и q в градусах)
{
      double fg = fmod(fi, 360.0);
      double ff = (fg / 180.0)*pi; // Перевод в радианы

      fg = fmod(q, 360.0);
      double qq = (fg / 180.0)*pi; // Перевод в радианы

      CMatrix VM(4, 4);

      VM(0, 0) = -sin(ff);            VM(0, 1) = cos(ff);
      VM(1, 0) = -cos(qq)*cos(ff);    VM(1, 1) = -cos(qq)*sin(ff);
      VM(1, 2) = sin(qq);
      VM(2, 0) = -sin(qq)*cos(ff);    VM(2, 1) = -sin(qq)*sin(ff);
      VM(2, 2) = -cos(qq);  VM(2, 3) = r;
      VM(3, 3) = 1;

      return VM;
}

//---------------------------------------------------------------------
---------------
CMatrix SphereToCart(CMatrix& PView)
// Преобразует сферические координаты PView  точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
```

```
// PView(2) - q - угол(отсчетот оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
{
	CMatrix R(3);

	double r = PView(0);
	double fi = PView(1);                              // Градусы
	double q = PView(2);                               // Градусы

	double fi_rad = (fi / 180.0)*pi;// Перевод fi в радианы
	double q_rad = (q / 180.0)*pi;        // Перевод q в радианы

	R(0) = r * sin(q_rad)*cos(fi_rad);    // x- координата точки
наблюдения
	R(1) = r * sin(q_rad)*sin(fi_rad);    // y- координата точки
наблюдения
	R(2) = r * cos(q_rad);                      // z- координата точки
наблюдения

	return R;
}

//-------------------------- GetProjection ---------------------------
----------------

void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD&
PR)
// Вычисляет координаты проекции охватывающего фигуру паралелепипеда
на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая
поверхность
// PView - координаты точки наблюдения в мировой сферической системе
координат
// PR - проекция
{
	double Zmax = Data.MaxElement();
	double Zmin = Data.MinElement();
	CMatrix PS(4, 4);     // Точки в мировом пространстве
	PS(3, 0) = 1; PS(3, 1) = 1;      PS(3, 2) = 1;    PS(3, 3) = 1;
	CMatrix MV = CreateViewCoord(PView(0), PView(1), PView(2));
	//Матрица(4x4) пересчета


	//в видовую систему координат
	PS(0, 0) = RS.left;
	PS(1, 0) = RS.top;
	PS(2, 0) = Zmax;
```

```
	PS(0, 1) = RS.left;
	PS(1, 1) = RS.bottom;
	PS(2, 1) = Zmax;

	PS(0, 2) = RS.right;
	PS(1, 2) = RS.top;
	PS(2, 2) = Zmax;

	PS(0, 3) = RS.right;
	PS(1, 3) = RS.bottom;
	PS(2, 3) = Zmax;

	CMatrix Q = MV * PS;	// Координаты верхней плоскости
параллелепипеда в видовой СК
	CMatrix V = Q.GetRow(0);	// Строка X - координат
	double Xmin = V.MinElement();
	double Xmax = V.MaxElement();
	V = Q.GetRow(1);	// Строка Y - координат
	double Ymax = V.MaxElement();

	PS(2, 0) = Zmin;
	PS(2, 1) = Zmin;
	PS(2, 2) = Zmin;
	PS(2, 3) = Zmin;

	Q = MV * PS;	// Координаты нижней плоскости
параллелепипеда в видовой СК
	V = Q.GetRow(1);	// Строка Y - координат
	double Ymin = V.MinElement();
	PR.SetRectD(Xmin, Ymax, Xmax, Ymin);
}
```

Реализация класса ChildView:

Первым делом следует создать карту сообщений, а также реализовать Конструкторы/Деструкторы класса.

```
#include "stdafx.h"
#include "Lab07.h"
#include "ChildView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Реализация карты сообщений.
// Подробнее - http://www.codenet.ru/progr/visualc/mfc/mfc3.php.
BEGIN_MESSAGE_MAP(CChildView, CWnd)
	ON_WM_PAINT()
	// Меню выбора команд.
	ON_COMMAND(ID_Pyramid_Color, &CChildView::OnPyramid)
	ON_WM_KEYDOWN()
	// Изменение окна.
	ON_WM_SIZE()
```

```
END_MESSAGE_MAP()

// Конструкторы/Деструкторы
CChildView::CChildView()
{
	Index = 0;		// начальный индекс
	PView.RedimMatrix(3);	// матрица для точки камеры для наблюдения
}
CChildView::~CChildView() { }

// Обработчики сообщений CChildView
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
	if (!CWnd::PreCreateWindow(cs))
		return FALSE;

	cs.dwExStyle |= WS_EX_CLIENTEDGE;
	cs.style &= ~WS_BORDER;
	cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
		::LoadCursor(nullptr, IDC_ARROW),
reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), nullptr);

	return TRUE;
}
```

Далее идёт реализация методов:

```
// Данная функция вызывается при каждом изменении окна.
void CChildView::OnSize(UINT nType, int cx, int cy)
{
	CWnd::OnSize(nType, cx, cy);			// для
динамического изменения окна
	WinRect.SetRect(50, 50, cx - 50, cy - 50); // параметры окна
рисования
}

void CChildView::OnPyramid()
{
	CFrameWnd* pWnd = GetParentFrame(); // Получаем родительское
рамочное окно.

	// Задаем начальную точку наблюдения в сферической системе
координат.
	// PView(0) - r.
	PView(0) = 10;
	// PView(1) - φ.
	PView(1) = 20;
	// PView(2) - θ.
	PView(2) = 110;

	Index = 4;
	Invalidate();
}
```

```
void CChildView::OnPaint()
{
	// Контекст устройства для рисования.
	CPaintDC dc(this);

	if (Index == 4)                         // для варианта с падением
света
		PIR.ColorDraw(                  // параметры для падения
цвета
			dc,                             // контекст рисования
			PView,                          // к-ты точки камеры
для наблюдения
			WinRect,                    // параметры окна рисования
			RGB(128, 236, 158));  // основной цвет пирамиды без
учета теней
}

void CChildView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
		if (Index == 4)
	{
		switch (nChar)
		{
		case VK_UP:
		{
			double d = PView(2) - 1;
			if (d >= 0)PView(2) = d;
			break;
		}
		case VK_DOWN:
		{
			double d = PView(2) + 1;
			if (d <= 180)PView(2) = d;
			break;
		}
		case VK_LEFT:
		{
			double d = PView(1) - 1;
			if (d >= -180)PView(1) = d;
			else PView(1) = d + 360;
			break;
		}
		case VK_RIGHT:
		{
			double d = PView(1) + 1;
			if (d <= 180)PView(1) = d;
			else PView(1) = d - 360;
			break;
		}
```

```
            }
            Invalidate();
      }
      CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

Конечным этапом становится реализация файлов, необходимого для корректного исполнения программы.

MainFrm.h

```
#pragma once
#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
      CMainFrame() noexcept;
protected:
      DECLARE_DYNAMIC(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределение
public:
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
AFX_CMDHANDLERINFO* pHandlerInfo);

// Реализация
public:
      virtual ~CMainFrame();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // встроенные члены панели элементов управления
      CStatusBar        m_wndStatusBar;
      CChildView    m_wndView;

// Созданные функции схемы сообщений
protected:
      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
      afx_msg void OnSetFocus(CWnd *pOldWnd);
      DECLARE_MESSAGE_MAP()
```

```
};
```

MainFrm.cpp

```
#include "stdafx.h"
#include "Lab07.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	ON_WM_CREATE()
	ON_WM_SETFOCUS()
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // индикатор строки состояния
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

// Создание или уничтожение CMainFrame

CMainFrame::CMainFrame() noexcept
{
	// TODO: добавьте код инициализации члена
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	//-----------------------------------------------------------------
----
	int width = 900, height = 680;
	MoveWindow((GetSystemMetrics(SM_CXSCREEN) / 2 - width / 2),
		(GetSystemMetrics(SM_CYSCREEN) / 2 - height / 2), width,
height);
	//-----------------------------------------------------------------
----
```

```
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	// создать представление для размещения рабочей области рамки
	if (!m_wndView.Create(nullptr, nullptr, AFX_WS_DEFAULT_VIEW,
CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, nullptr))
	{
		TRACE0("Не удалось создать окно представлений\n");
		return -1;
	}

	if (!m_wndStatusBar.Create(this))
	{
		TRACE0("Не удалось создать строку состояния\n");
		return -1;	// не удалось создать
	}
	m_wndStatusBar.SetIndicators(indicators,
sizeof(indicators)/sizeof(UINT));

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: изменить класс Window или стили посредством изменения
	//  CREATESTRUCT cs

	cs.style = WS_OVERLAPPED | WS_CAPTION | FWS_ADDTOTITLE
		| WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX;

	cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
	cs.lpszClass = AfxRegisterWndClass(0);
	return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}
```

```
#endif //_DEBUG


// Обработчики сообщений CMainFrame

void CMainFrame::OnSetFocus(CWnd* /*pOldWnd*/)
{
	// передача фокуса окну представления
	m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
AFX_CMDHANDLERINFO* pHandlerInfo)
{
	// разрешить ошибки в представлении при выполнении команды
	if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
		return TRUE;

	// в противном случае выполняется обработка по умолчанию
	return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

Stdafx.h

```
// stdafx.h: включите файл для добавления стандартных системных файлов
//или конкретных файлов проектов, часто используемых,
// но редко изменяемых

#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Исключите редко используемые
компоненты из заголовков Windows
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // некоторые
конструкторы CString будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых
предупреждений MFC
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // основные и стандартные компоненты MFC
#include <afxext.h>         // расширения MFC

#include <FLOAT.H> // Для  DBL_MAX , DBL_MIN
#include <fstream>
#include <math.h>
#include "CMatrix.h"
```

```
#include <vector>

#include "LibGraph.h"
#include "LibPyramid.h"

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // поддержка MFC для типовых элементов
управления Internet Explorer 4
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // поддержка MFC для типовых элементов
управления Windows
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>     // поддержка MFC для лент и панелей
управления

#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='x86' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#elif defined _M_X64
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#else
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#endif
#endif
```

Resource.h

```
//{{NO_DEPENDENCIES}}
// Включаемый файл, созданный в Microsoft Visual C++.
// Используется в Lab03.rc
//
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED                 100
#define IDR_MAINFRAME                   128
#define IDR_Lab01TYPE                   130
#define ID_Pyramid_Color                32771

// Следующие стандартные значения для новых объектов
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
```

```
#define _APS_NEXT_RESOURCE_VALUE        312
#define _APS_NEXT_COMMAND_VALUE         32776
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           310
#endif
#endif
```

targetver.h

```
#pragma once

// Включение SDKDDKVer.h обеспечивает определение самой последней
доступной платформы Windows.

// Если требуется выполнить сборку приложения для предыдущей версии
Windows, включите WinSDKVer.h и
// задайте для макроопределения _WIN32_WINNT значение поддерживаемой
платформы перед вхождением SDKDDKVer.h.

#include <SDKDDKVer.h>
```

Lab07.h:

```
// Lab07.h: основной файл заголовка для приложения Lab07
//
#pragma once

#ifndef __AFXWIN_H__
     #error "включить stdafx.h до включения этого файла в PCH"
#endif

#include "resource.h"       // основные символы


// CLab01App:
// Сведения о реализации этого класса: Lab01.cpp
//

class CLab01App : public CWinApp
{
public:
     CLab01App() noexcept;

// Переопределение
public:
     virtual BOOL InitInstance();
     virtual int ExitInstance();

// Реализация

public:
     afx_msg void OnAppAbout();
```

```
	DECLARE_MESSAGE_MAP()
};

extern CLab01App theApp;
```

Lab07.cpp:

```
// Lab02.cpp: определяет поведение классов для приложения.
//

#include "stdafx.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "Lab07.h"
#include "MainFrm.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CLab01App

BEGIN_MESSAGE_MAP(CLab01App, CWinApp)
END_MESSAGE_MAP()

// Создание CLab01App

CLab01App::CLab01App() noexcept
{

	SetAppID(_T("Lab01.AppID.NoVersion"));
}

// Единственный объект CLab01App

CLab01App theApp;


// Инициализация CLab01App

BOOL CLab01App::InitInstance()
{
	// InitCommonControlsEx() требуются для Windows XP, если манифест
	// приложения использует ComCtl32.dll версии 6 или более поздней
версии для включения
	// стилей отображения.  В противном случае будет возникать сбой
при создании любого окна.
	INITCOMMONCONTROLSEX InitCtrls;
```

```
	InitCtrls.dwSize = sizeof(InitCtrls);
	// Выберите этот параметр для включения всех общих классов
управления, которые необходимо использовать
	// в вашем приложении.
	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);

	CWinApp::InitInstance();


	// Инициализация библиотек OLE
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
		return FALSE;
	}

	AfxEnableControlContainer();

	EnableTaskbarInteraction(FALSE);

	// Для использования элемента управления RichEdit требуется метод
AfxInitRichEdit2()
	// AfxInitRichEdit2();

	// Стандартная инициализация
	// Если эти возможности не используются и необходимо уменьшить
размер
	// конечного исполняемого файла, необходимо удалить из следующего
	// конкретные процедуры инициализации, которые не требуются
	// Измените раздел реестра, в котором хранятся параметры
	// TODO: следует изменить эту строку на что-нибудь подходящее,
	// например на название организации
	SetRegistryKey(_T("Локальные приложения, созданные с помощью
мастера приложений"));


	// Чтобы создать главное окно, этот код создает новый объект окна
	// рамки, а затем задает его как объект основного окна приложения
	CFrameWnd* pFrame = new CMainFrame;
	if (!pFrame)
		return FALSE;
	m_pMainWnd = pFrame;
	// создайте и загрузите рамку с его ресурсами
	pFrame->LoadFrame(IDR_MAINFRAME,
		WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, nullptr,
		nullptr);


	// Разрешить использование расширенных символов в горячих
```

```
клавишах меню
	CMFCToolBar::m_bExtCharTranslation = TRUE;
	// Одно и только одно окно было инициализировано, поэтому
отобразите и обновите его
	pFrame->ShowWindow(SW_SHOW);
	pFrame->UpdateWindow();
	return TRUE;
}

int CLab01App::ExitInstance()
{
	//TODO: обработайте дополнительные ресурсы, которые могли быть
добавлены
	AfxOleTerm(FALSE);

	return CWinApp::ExitInstance();
}

// Обработчики сообщений CLab01App


// Диалоговое окно CAboutDlg используется для описания сведений о
приложении

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg() noexcept;

// Данные диалогового окна
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_ABOUTBOX };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // поддержка
DDX/DDV

// Реализация
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
{
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// Команда приложения для запуска диалога
void CLab01App::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

Литература:

1. Файлик с построением Пирамиды с удалением НГ
2. Этот сайтик http://www.codenet.ru/progr/visualc/mfc/
3. Основы КГИГ и графики NEW